E-Mail-Integration in eDemocracy-Systeme

Bachelorarbeit

von

Denis Klaus Küchemann

aus

Wuppertal

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

August 2013

Betreuer:
Philipp Hagemeister M. Sc.

Danksagung

An dieser Stelle möchte ich meiner Familie und meiner Freundin dafür danken, dass sie mich während des Anfertigens der Arbeit in jeder Lebenslage unterstützt haben.

Florian Koch danke ich für sein Engagement beim Korrekturlesen meiner Arbeit.

Mein besonderer Dank gilt Philipp Hagemeister für die zu jedem Zeitpunkt fachlich exzellente und persönlich motivierende Betreuung meiner Arbeit.

Im Speziellen möchte ich dem Mathematiker Sebastian Welsch danken, der mich während des Studiums zielgerichtet begleitet hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In Zeiten sogenannter Politikverdrossenheit kann direkt praktizierte Bürgerbeteiligung an gesellschaftlichen Entscheidungsprozessen durch den Einsatz moderner Web-Technologien wiederbelebt bzw. ganz neu definiert werden.

Die Web-Applikation *adhocracy*, eine Online-Beteiligungs-Plattform für politische Diskurse, bedient sich des Ansatzes, politische Diskussionen bzw. das Abstimmen über diesbezogene Vorschläge zu veröffentlichen.

Neue Beiträge innerhalb eines Vorschlags können bis dato über die Web-Oberfläche von adhocracy verfasst werden.

Durch die allgemeine Zugänglichkeit von E-Mail bietet es sich an, ein direktes Kommentieren per E-Mail zu ermöglichen. Durch die Plattformunabhängigkeit von E-Mail und das Umgehen von möglichen Browser-Inkompatibilitäten kann eine schnelle und effektive Beteiligung des Benutzers am System gefördert werden.

Insbesondere durch den heutzutage gewohnten Umgang mit mobiler Kommunikation ist eine Benutzung von E-Mails für direkte Antworten geeignet. Die Benutzerfreundlichkeit kann auf Mobilgeräten so enorm erhöht werden. Das Benutzen eines Browsers

auf Mobilgeräten hingegen kann durch eine langsame Internetverbindung, kleine Displays oder durch oben genannte Browser-Inkompatibilitäten schwierig sein. Man kann dadurch, dass E-Mails vermutlich regelmäßig überprüft werden und kein Web-Browser mehr benutzt werden muss, eine regere Beteiligung am System erhoffen.

Das weltweit größte soziale Netzwerk Facebook besitzt ebenfalls eine Funktion, mit der man über E-Mail auf Kommentare antworten kann. Allerdings unterliegt die E-Mail-Kommentarfunktion von Facebook diversen Einschränkungen. So ist es z.B. nicht möglich, Bilder anzuhängen oder über HTML zu verlinken, einen Kommentar zu bewerten oder eine beliebige E-Mail-Adresse zum Kommentieren zu benutzen.

1.2 Problemstellung

Im Rahmen dieser Bachelorarbeit soll untersucht werden, inwiefern E-Mail-Funktionen in eDemocracy-Systeme integriert oder erweitert werden können. Es soll über eine E-Mail-Antwort ermöglicht werden, bestehende Vorschläge und Kommentare ohne Anmeldung auf der Web-Oberfläche zu kommentieren, bzw. ein Votum abzugeben. Dabei soll beachtet werden, dass die Systemsicherheit nicht kompromittiert werden kann. Ferner sollte ein einfaches E-Mail-Parsing integriert werden, welches Text- und PGP-Signaturen entfernt. Des Weiteren soll erklärt werden, ob und wie der Mail Transfer Agent (MTA) eingerichtet werden muss.

1.3 Struktur der Arbeit

Kapitel 2 beschäftigt sich mit den Komponenten, die erforderlich sind, um eine effiziente Implementation in adhocracy vorzunehmen. Hier wird das Design des Programms ausgearbeitet und die jeweilige Implementation erläutert. In Kapitel 3 wird ein Resümee gezogen und abschließend in Kapitel 4 ein Ausblick gegeben.

Kapitel 2

Programm-Design

Die eDemocracy-Plattform *adhocracy* wurde im Web-Framework Pylons erstellt, welches in der Programmiersprache Python geschrieben ist. Pylons stellt bereits viele Funktionen über WSGI-Middleware[1] zur Verfügung.

Je nach Einstellung der Benutzerrechte einer Instanz (das Hauptthema, in dem die jeweiligen Vorschläge gespeichert sind), muss ein Benutzer sich zuerst authentifizieren, um einen Kommentar zu verfassen. Eine Instanz bzw. die Gruppe der anonymen Benutzer kann so eingestellt sein, dass auch anonyme Kommentare innerhalb eines Vorschlags verfasst werden dürfen (Autorisierung).

Kommentare werden in einer relationalen Datenbank gespeichert.

Ein Worker-Prozess übernimmt Aufgaben, die länger andauern können und führt diese asynchron im Hintergrund aus. Dazu gehört das Erstellen von Events. Unter anderem wird der Benachrichtigungsvorgang über neue Kommentare eingeleitet, sobald ein Event über einen neuen Kommentar erstellt wurde.

Ein MTA muss bereits zumindest so eingerichtet sein, dass der Benutzer nach einer Registrierung seine E-Mail-Adresse bestätigen kann und E-Mail-Benachrichtigungen über neue Kommentare bekommt.

[1] Web-Server-Gateway-Interface

Für E-Mail-Kommentare müssen einige Mechanismen anders benutzt werden und Funktionen hinzugefügt werden. In Abbildung 2.1 ist der derzeitige Stand vor einer Implementierung einer E-Mail-Kommentarfunktion abgebildet: Möchte ein Benutzer einen Kommentar verfassen, meldet er sich zuerst über einen Web-Browser (Authentifizierung) bei adhocracy an. Die Autorisierungmechanismen prüfen, ob der Benutzer die Berechtigungen besitzt, den Kommentar zu verfassen. Falls dem so ist, wird der Kommentar in die Datenbank geschrieben. Alle anderen Benutzer, die den Vorschlag abonniert haben, erhalten eine E-Mail-Benachrichtigung über den neuen Kommentar.

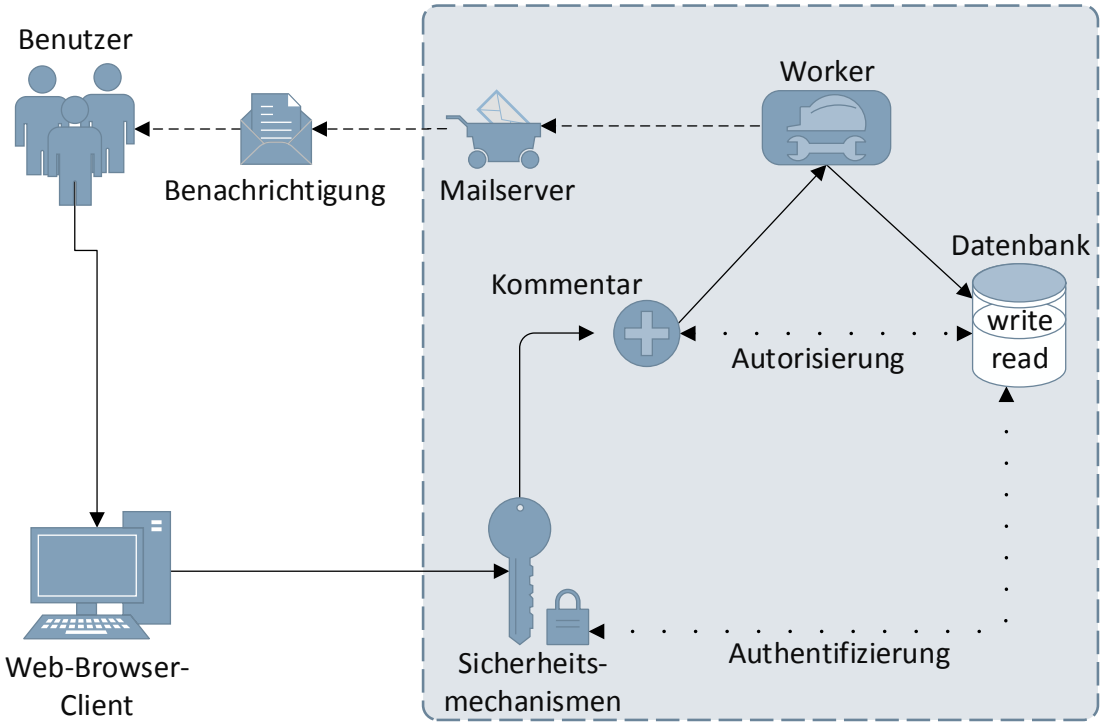

Abbildung 2.1: Aktueller Stand: Schreiben eines Kommentars in adhocracy

2.1 Grundlegender Aufbau

Zuerst wird untersucht, ob Sicherheitsmechanismen für die neuen Anforderungen wiederverwendet werden können oder abgeändert werden müssten, um für E-Mail-Kommentare eine mindestens ebenso hohe Sicherheit gewährleisten zu können (siehe Kapitel 2.2).

Es ist möglich, dass der MTA weiter konfiguriert werden muss (siehe Kapitel 2.3).

Der Kommentiervorgang wird in einem Worker initialisiert bzw. dort ausgeführt (siehe Kapitel 2.4).

Ankommende E-Mails mit Kommentarantworten sollen in Echtzeit abgeholt werden können (siehe Kapitel 2.5).

Diese E-Mails müssen vor dem Schreiben in die Datenbank analysiert werden (Parsing) (siehe Kapitel 2.6).

Zum Speichern der E-Mail-Kommentare wird mit der Datenbank kommuniziert, in der auch alle durch die Web-Oberfläche erstellten Benutzer und Kommentare gespeichert sind (siehe Kapitel 2.7).

Abbildung 2.2 zeigt, wie ein Kommentar mit E-Mails verfasst werden soll: Der Benutzer versendet eine E-Mail-Antwort auf die Benachrichtigung über einen neuen Kommentar. Nachdem der MTA die E-Mail empfangen hat, wird die E-Mail abgeholt und geparsed - beim Parsing findet auch die Authentifizierung statt. Ist der Benutzer autorisiert, einen Kommentar zu verfassen, wird der Kommentar in die Datenbank geschrieben und es wird wieder eine E-Mail-Benachrichtigung über einen neuen Kommentar versendet. Andernfalls erhält der Benutzer eine E-Mail darüber, dass kein Kommentar verfasst werden konnte.

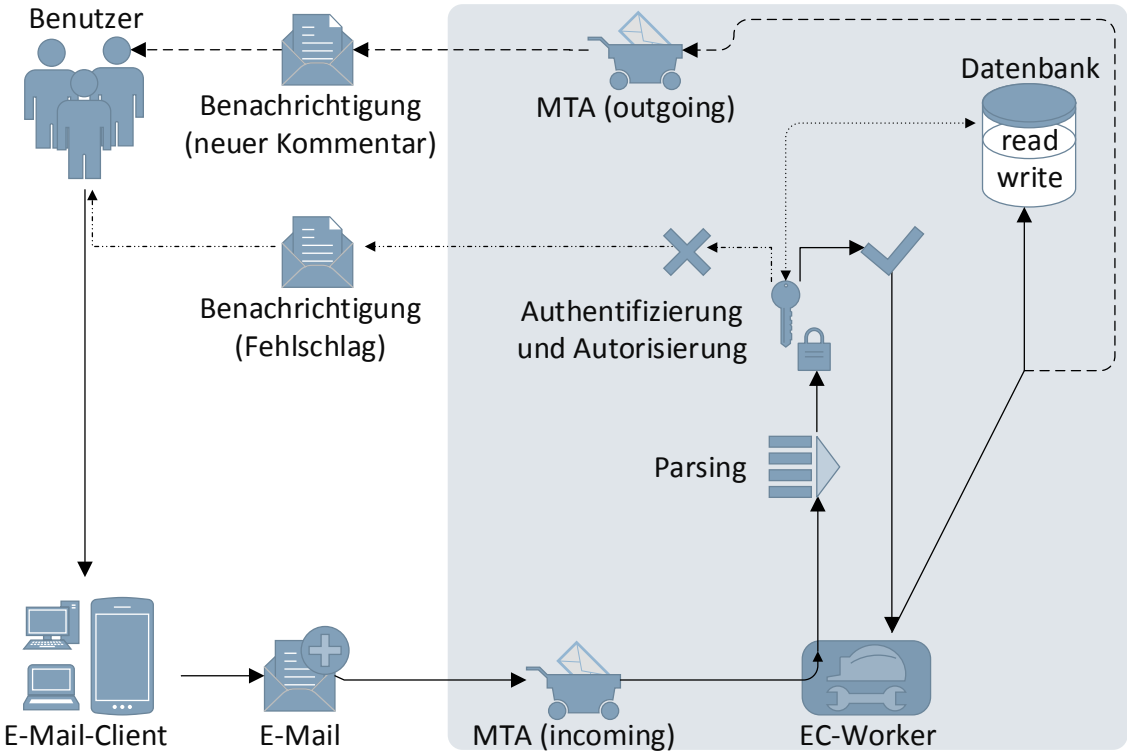

Abbildung 2.2: Nach E-Mail-Integration: Schreiben eines E-Mail-Kommentars

2.2 Sicherheit

Es ist nötig, zu kontrollieren, ob ein Benutzer eine Kommentarantwort verfassen darf. Dabei muss nicht nur beachtet werden, wer eine Kommentarantwort verfasst sondern auch wann das Kommentieren erlaubt ist.

Da auch bereits das Kommentieren über die Web-Oberfläche von Sicherheitsmechanismen überprüft wird, ist es sinnvoll, diese Konzepte auch auf E-Mail-Kommentare zu übertragen. Allerdings muss sich der Benutzer anders authentifizieren, da keine Web-Oberfläche benutzt wird, über die sich der Benutzer anmelden kann. Die bereits vorhandenen Autorisierungsmechanismen werden benutzt, um festzustellen, inwiefern der Benutzer berechtigt ist, eine Antwort zu verfassen.

2.2.1 Design

Eine Authentifizierung sollte unabhängig von der Absender-Adresse des Benutzers erfolgen können. Es kann vorkommen, dass ein Benutzer mit einem Mail User Agent (MUA) eine andere E-Mail-Adresse benutzt als die in der Datenbank verknüpfte E-Mail-Adresse zum Benutzerkonto. Außerdem verhindert eine Unabhängigkeit von der E-Mail-Adresse sogenanntes Spoofing[2], durch das man sich für einen anderen Benutzer ausgeben könnte, um in dessen Namen Kommentarantworten zu schreiben.

Authentifizierung

Es wäre möglich, dem Benutzer eine vorformulierte Klartext-E-Mail für seine Antwort zur Verfügung zu stellen, in welche die benötigten Daten bereits eingetragen sind. Die Gefahr, dass der Benutzer diese aber abändern oder nicht richtig benutzen würde, wäre sehr hoch. Das könnte eine Authentifizierung außerordentlich schwierig oder unmöglich gestalten, da der Benutzer nicht von seiner Absender-Adresse abhängig sein soll, ebenso wenig aber mit langen Erklärungen verwirrt werden soll.
Eine Übergabe über das Subjekt der E-Mail birgt geringere Probleme, dennoch ist die Gefahr als zu hoch einzuschätzen, dass das Subjekt abgeändert und keine Authentifizierung vorgenommen werden könnte.

Ebenso wäre es denkbar, dass der Benutzer eine HTML-E-Mail erhält und die Übergabe der benötigten Daten über eine Verbindung mit dem Server durchgeführt wird. Unter der Annahme, dass jeder Benutzer Klartext-E-Mails empfangen kann aber nur ein geringer Teil HTML-E-Mails empfangen kann oder möchte, kann eine derartige Implementation nur zusätzlich, nicht aber als Ersatz erfolgen. Da z.B. ausführbarer Code in E-Mails ein potentielles Risiko darstellt, kann man nicht davon ausgehen, dass diese Möglichkeit nutzbar ist, sodass HTML-E-Mails nur dekorativen Nutzen hätten.

Sinnvoll erscheint eine Übergabe direkt über die E-Mail-Adresse, auf die geantwortet werden soll. So können Subjekt und der Payload der E-Mail beliebig bearbeitet werden und es kann intuitiv geantwortet werden.

[2] Mail-Spoofing bezeichnet Täuschungsversuche mit gefälschten E-Mail-Adressen.

Es gibt mehrere Möglichkeiten, eine sichere Datenübergabe zu gewährleisten. Dabei gilt zu beachten, dass bei der Übergabe über die E-Mail-Adresse deren lokaler Teil[3] eine maximale Länge von 64 Zeichen hat und diese nicht überschritten werden sollen [Kle08]. Es muss ein Wert generiert werden, der eindeutig nachweisen kann, um welchen Benutzer es sich handelt und der nicht repliziert werden kann.

- **Hashing** kann benutzt werden, um eine Signatur zu erstellen. Für die Generierung eines nicht zu replizierenden Wertes wird das zufällig generierte adhocracy-Crypto-Secret[4] benutzt. Dazu müssen allerdings weitere Referenzen für einen späteren Vergleich des Hashes hinzugefügt werden, weil es nicht möglich ist, einen Hash zu entschlüsseln. Diese Referenzen werden nicht mitgehasht und demnach als Klartext übergeben.
  Als eindeutige Referenz eignet sich erwartungsgemäß die Benutzer-ID. Um zuzuordnen, auf welchen Kommentar der Benutzer antworten möchte, muss ebenfalls die Kommentar-ID übergeben werden. Die Autorisierungsfunktionen prüfen anhand dessen die benötigten Berechtigungen des Benutzers. Benutzer- und Kommentar-ID werden aneinandergereiht und mitsamt des Crypto-Secrets gehasht. Das dadurch erstellte Sicherheitstoken kann in Verbindung mit den in Klartext übergebenen Referenzen den Benutzer sicher authentifizieren und autorisieren.

- **Verschlüsselungen** mit einem Schlüsselpaar hätten den Vorteil, bei einer Übergabe nicht mehr Klartext-IDs übergeben zu müssen. Bei einer Antwort müssten die mit dem öffentlichen Schlüssel verschlüsselten Übergabewerte mit dem privaten Schlüssel wieder entschlüsselt werden. Das Verschlüsselungsverfahren ist allerdings in hiesigem Fall unbrauchbar, da sehr schnell die maximal erlaubte Länge des lokalen Teils der E-Mail-Adresse überschritten werden kann. Eine Signatur mit Hashing hat darüber hinaus den Vorteil, dass keine zusätzlichen Schlüssel(paare) zum Crypto-Secret benötigt werden.

- **Noncen** hätten zwar den Vorteil, dass man Kommentar-Spamming einfach verhindern kann, da dann nur einmal eine Antwort auf den Kommentar erstellt werden kann, dies wäre aber natürlich nicht wünschenswert. Eine zufällig generierte

[3] Der lokale Teil der E-Mail-Adresse bezeichnet den Teil vor dem letzten „@“.
[4] Das adhocracy-Crypto-Secret stammt aus der Konfigurationsdatei des Web-Frameworks

Nonce hätte keine Notwendigkeit weiterer Parameterübergaben und müsste nicht verschlüsselt oder gehasht werden. Allerdings müsste dafür die Datenbank beansprucht werden, was weitaus aufwändiger wäre als das simple Hashing, was auch die Nonce zu keiner echten Alternative macht.

Autorisierung

Der Benutzer kann sich in verschiedenen Gruppen mit verschiedenen Berechtigungen befinden. Dort ist z.B. gespeichert, ob der Benutzer in dieser Gruppe Kommentare verfassen darf. Die bereits vorhandenen Mechanismen überprüfen anhand der Gruppenberechtigungen des Benutzers, ob er in der aktiven Instanz Kommentare verfassen darf oder der Kommentar, auf den geantwortet werden soll, gelöscht wurde. Weiterhin darf die Instanz nicht eingefroren sein. Ebenso wird überprüft, ob der Benutzer seine E-Mail-Adresse bestätigt hat.
Der Autorisierungsmechanismus überprüft ebenfalls, ob der Benutzer berechtigt ist, ein Votum abzugeben. Fehlt dem Benutzer die Berechtigung, kann er zwar keine Stimme mehr für oder gegen einen Kommentar bzw. einen Vorschlag abgeben - allerdings hat diese Überprüfung keinen Einfluss auf die Berechtigung, eine Kommentarantwort erstellen zu können. Anonymen Benutzern ist es je nach Einstellungen in einer Instanz möglich, Kommentare zu verfassen. Evtl. möchte man aber vermeiden, dass nicht registrierte Benutzer zu einem Vorschlag auch abstimmen können. Ebenso ist es denkbar, dass es bei Umfragen nur registrierten Benutzern gestattet sein soll, einen Kommentar zu verfassen. Aus diesen Gründen werden die Kommentar-Berechtigungen und die vote-Berechtigungen getrennt.

Sichere Abholung von E-Mails

Für die Verwendung von IMAP (siehe Kapitel 2.5) wird SSL als Sicherheitsbedingung vorausgesetzt. SSL ist ein Verschlüsselungsprotokoll und Standard für sicherere Dateiübertragung im Internet. SSL bietet Transportschichtsicherheit für TCP-Anwendungen über SSL-Dienste und soll Lauschangriffe, Manipulation und Fälschung von Nachrich-

ten verhindern [FKK11]. Verbindungsversuche zu IMAP-Servern, welche kein SSL unterstützen, werden abgebrochen.

2.2.2 Implementation

Eine Implementierung erfolgt, wegen der genannten Vorteile, über die E-Mail-Adresse. Um die Authentifizierung sicher zu gestalten, wird Hashing benutzt.

Adhocracy besitzt bereits Funktionen, die hashen können. Durch die Beschränkung der Länge des lokalen Teils der E-Mail-Adresse muss darauf geachtet werden, dass ein Hash nicht zu lang wird. Da bei einer Änderung dieser Funktion nicht mehr gewährleistet werden kann, dass ein Hash kurz genug ist, werden die vorhandenen Hashingfunktionen von adhocracy nicht benutzt. Eine Implementierung wird deswegen unabhängig von anderen Funktionen vorgenommen.
Das Hashing benutzt den SHA1-Algorithmus, der einen genau 40 Zeichen langen Hash generiert. Ebenso könnte man MD5-Hashing benutzen, welches nur einen 32 Zeichen langen Hash zurückgeben würde. Dies wäre zwar für die Länge des lokalen Teils der E-Mail-Adresse zuträglich aber ein Hashing mit MD5 [Kam12] wird allgemein als nicht mehr sicher genug eingestuft.

Die E-Mail-Benachrichtigung, die der Benutzer bekommt, wenn er einem Vorschlag folgt, beinhaltet weiterhin die Absender-Adresse, von der alle E-Mails verschickt werden. Bei einer Kommentarantwort wird allerdings eine generierte E-Mail-Adresse der Form
Name <subs.benutzerid-kommentarid.sicherheitstoken@domain.tld> benutzt. Möglich ist dies durch das Hinzufügen des E-Mail-Headers `In-Reply-To` [Res01]. „Name“ wird ersetzt und dazu benutzt, um darauf hinzuweisen, dass auf diese E-Mail geantwortet werden kann. Zugleich sollte auch die E-Mail-Adresse selbst für Benutzer und Empfänger einen Hinweis darauf enthalten, da derart lange, aus Zeichenfolgen bestehende, E-Mail-Adressen ungewöhnlich sind. Dazu wird `subs.` als Präfix für die Zeichenfolge benutzt. „subs“ ist die Kurzform für „subscription“. Für Benutzer- und Kommentar-ID bleiben, durch die Beschränkung auf 64 Zeichen des lokalen Teils (Siehe Kapitel 2.2.1), mehr Zeichen (17) bei der Wahl dieser Kurzform. Andernfalls wären es nur neun Zeichen.

So wird gewährleistet, dass möglichst viele registrierte Benutzer bei möglichst vielen Kommentaren den Dienst nutzen können. So ist es beispielsweise möglich, dass 10.000 Benutzer fast eine Billionen Kommentare verfassen könnten. Bei nur neun freien Zeichen wären bei 10.000 Benutzern nur fast 10.000 Kommentare möglich.

2.3 MTA

Der MTA ist notwendig, um Benachrichtigungen über neue Kommentare zu verschicken und E-Mail-Kommentarantworten zu empfangen. Wünschenswert ist die Benutzung eines beliebigen MTAs, welcher nur minimal eingerichtet werden muss.
Das Sicherheitsmodell gibt vor, eine Authentifizierung über die E-Mail-Adresse zu erledigen. Dafür muss es möglich sein, E-Mails an die für den Benutzer generierte E-Mail-Adresse zurückschicken zu können. Dies kann durch einen Alias-Verweis erreicht werden.

2.3.1 Design

Oft bedienen sich MTAs der Unix-Alias-Datei, während andere eigene Alias-Dateien verwalten. Diese müssen über die spezifische Konfiguration des MTAs eingerichtet werden.

Es können reguläre Ausdrücke benutzt werden (siehe Kapitel 2.2.2), um E-Mails weiterzuleiten. Der benutzte MTA muss dazu so eingerichtet werden, dass er reguläre Ausdrücke auch unterstützt bzw. lesen kann. Im Normalfall reicht es nicht aus, nur die Alias-Datei(en) anzupassen.
Alternativ kann eine Wildcard benutzt werden, um E-Mails weiterzuleiten. Es werden dann zwar auch E-Mails weitergeleitet, die nicht die gewünschte Form der E-Mail-Adresse haben aber die Konfiguration des MTAs kann einfacher werden.

Bisher muss der MTA kein eingerichtetes Postfach haben, da E-Mails nur verschickt, nicht aber empfangen werden müssen. Manche MTAs benötigen kein Postfach für das

Empfangen von E-Mails. Für eine Überwachung von lokalen Ordnern (siehe Kapitel 2.5) ist es allerdings Voraussetzung, dass so ein Postfach existiert.

Konfiguration

Adhocracy kann bisher auf einigen unixoiden Betriebssystemen ausgeführt werden. Für die Überwachung von lokalen Ordnern wird demnach eine unixoide Ordnerstruktur angenommen. Anhand des Unix-Benutzernamens wird ermittelt, wo sich die lokalen Ordner befinden. An diesen Unix-Benutzer sollen die E-Mails durch den Alias-Verweis weitergeleitet werden.[5] Dazu ist es nötig, dass dieser Unix-Benutzer existiert.

Der reguläre Ausdruck für den lokalen Teil der E-Mail-Adresse, der benutzt wird, um die entsprechenden E-Mails zu empfangen, muss folgendermaßen in die Alias-Datei eingetragen werden:
`subs\.[1-9][0-9]*-[1-9][0-9]*\.[a-fA-F0-9]{40}`
Dieser reguläre Ausdruck muss vor der E-Mail-Adresse stehen, an die weitergeleitet werden soll.
Möchte man stattdessen eine Wildcard benutzen, um E-Mail-Kommentare weiterzuleiten, ändert man den Teil vor der Weiterleitungsadresse:
`subs.*`

2.4 Worker

Der Worker ist ein Prozess, der Hintergrunddaten verarbeitet. Er stellt eine Warteschlange zur Verfügung, die diese Aufgabe übernimmt. Die Warteschlange des vorhandenen Workers arbeitet die Aufträge der Web-Browser-Oberfläche ab.

[5] In der Konfiguration des Web-Frameworks muss dieser Benutzer eingetragen werden.

2.4.1 Design

Es bieten sich mehrere Möglichkeiten an, eine Implementation der E-Mail-Kommentar-Funktion vorzunehmen.

Da viele der vom Webframework benötigten Funktionen thread-lokal sind, bietet es sich an, einen eigenständigen Worker zu benutzen.

Ein **neuer Worker**, der sich nur um E-Mail-Antworten kümmern soll, kann bei einer hohen Last des bereits vorhandenen Workers sehr schnell die E-Mail-Kommentare bearbeiten. Der ursprüngliche Worker benutzt die Warteschlange *Redis Queue*. Redis Queue benötigt den Dienst *Redis*, eine nicht relationale und dadurch sehr schnelle Datenbank. Nach dem FIFO-Prinzip werden Referenzen zu Funktionen und deren Argumente in die Redis-Datenbank geladen. Diese Funktionen werden ausgeführt, sobald sie an der Reihe sind. Im Vergleich zu einer normalen Warteschlange ist es sinnvoll, Redis zu benutzen, wenn man viele verschiedene Aufgaben delegieren möchte. Eine normale Warteschlange ist lediglich in der Lage, Variablen zu übergeben. Funktionen, die dann ausgeführt werden sollen, stehen demnach bereits fest. Für einen Worker, der nur eine Aufgabe hat, reicht es hingegen aus, nur eine einfache Warteschlange zu benutzen. Da der neue Worker nur die Aufgabe hat, E-Mail-Kommentare zu bearbeiten, ist eine Verwendung von Redis nicht nötig. Redis Queue hat außerdem zwar keine Speicherlecks, kann dadurch aber langsam werden [Dri13]. Die Verwendung einer gewöhnlichen Warteschlange scheint hier zweckmäßiger zu sein. Mit einem zusätzlichen Worker kann man die E-Mail-Kommentar-Funktion komfortabel deaktivieren, ohne andere Dienste stoppen zu müssen bzw. das restliche System anzuhalten oder dessen Konfiguration zu ändern. Dadurch wäre z.B. die Möglichkeit gegeben, Wartungsarbeiten an Mailboxen vorzunehmen.

Mit **Threading** ist es möglich, der Einrichtung eines zusätzlichen Workers zu entgehen. Da die Warteschlange im Worker weitere Code-Ausführung verhindert, muss eine Implementation vorher geschehen, wenn eine Programminitialisierung im Worker stattfinden soll. Eine Implementation im Worker ist sinnvoll, da dieser ohnehin für die Verarbeitung von Hintergrunddaten gestartet werden muss. Die Konfiguration muss hier allerdings genauer betrachtet werden. Dadurch, dass diese immer thread-lokal ist, muss sie entweder in jedem Thread neu geladen werden oder kopiert und für den Thread eingerichtet

bzw. aktualisiert werden. Ein Neuladen der Konfiguration hat allerdings zur Folge, dass der Cache von *memcached* geleert wird. Memcached ist ein Dienst, der Teile der Web-Seite zwischenspeichert. Ein Leeren des Caches ist generell nicht erwünscht. Durch die Komplexität des Lade-Vorgangs wirkte das Kopieren der Konfiguration aus dem Worker sinnvoller aber „unsauber“. Ebenso war es überhaupt problematisch, eine thread-lokale Variable zu kopieren.

Beim Threading gibt es zwei Möglichkeiten, ankommende E-Mails zu bearbeiten.

- Einerseits kann, sobald eine neue E-Mail eingeht, ein **neuer Thread** gestartet werden. Dieser führt dann asynchron die zu erledigenden Aufgaben durch, während direkt weiter auf neue E-Mails gewartet wird (siehe Abschnitt 2.6). Allerdings traten beim Threading Probleme auf, deren Behebung die ohnehin, im Vergleich zum eigenständigen Worker, ausgedehnte Implementation weiter verkomplizierten. Ein eigenständiger Worker wirkt hier „sauberer“ als eine Umgehung von vielen kleinen Problemen durch Notlösungen.

- Andererseits kann **Redis Queue** benutzt werden, um einen Auftrag auszuführen. Das Hinzufügen eines Auftrages in die Redis-Warteschlange blockiert allerdings den aktiven Thread, da der Auftrag synchron ausgeführt wurde. Es war zum Zeitpunkt der Implementation nicht möglich, ohne weitere Einstellungen bzw. das Zurückspringen in den Haupt- oder in einen neuen Thread, die Redis-Warteschlange sinnvoll zu benutzen. Durch das synchrone Ausführen der Aufträge setzten die Überwachungsvorgänge aus. In dieser Zeit konnten keine neuen E-Mails erkannt werden, was diese Alternative zur Benutzung ausschließt und die Implementation eines eigenständiges Workers zweckmäßiger erscheinen lässt.

2.4.2 Implementation

Obwohl Threading oberflächlich betrachtet eine ebenso gute Alternative zu sein scheint, ist die Benutzung eines neuen Workers im Vergleich einfach und fehlerlos zu implementieren. Die Konfiguration des Web-Frameworks kann ohne Kopier- oder Ladevorgänge benutzt werden und auch Verbindungsabbrüche, die bei der Threading-Alternative mit

der Datenbank auftraten, sind so behoben worden. Wie gewünscht kann eine Warteschlange im Worker-Prozess benutzt werden, was mit der Benutzung von Redis Queue nicht ohne Weiteres möglich gewesen ist.

Sobald der Worker startet, wird anhand der Konfiguration ermittelt, ob entweder die lokalen Ordner oder ein IMAP-Server überwacht werden sollen. Nun wird auch die Datenbank-Session eingerichtet. Bei einer lokalen Überwachung werden die Ordnerstrukturen angelegt, sofern diese noch nicht existieren. Des Weiteren wird überprüft, ob Inotify vorhanden ist, welches allerdings unter Linux seit 2005 fester Bestandteil des Linux-Kernels ist. Falls die Ordner nicht existieren und nicht angelegt werden konnten oder Inotify nicht vorhanden ist, wird eine Fehlermeldung in der Log des Workers ausgegeben und die lokale Überwachung nicht gestartet. Zuletzt wird die Warteschlange eingerichtet, die auf neue Aufträge wartet.

2.5 Empfang von E-Mails

Es erscheint sinnvoll, den Empfang von E-Mails so zu implementieren, dass eine Überwachung von Postfächern in Echtzeit erfolgt. Eine Echtzeitüberwachung für ankommende E-Mails gewährleistet, dass Kommentare zeitnah bearbeitet werden können. Ein Benutzer kann sich somit zeitnah und aktiv an einem Vorschlag beteiligen.
Verschiedene Technologien ermöglichen es, diese Echtzeitüberwachung von E-Mails und E-Mail-Ordnern zu übernehmen. Im Folgenden werden die beiden unterschiedlichen Ansätze IMAP- und lokale Überwachung näher betrachtet und die Unterschiede erläutert.

Lokale Überwachung

Eine Option für die lokale Überwachung ist insbesondere sinnvoll für Intranet-Web-Oberflächen, ebenso aber für lokal laufende IMAP-Server. Die Standard-Unix-Mailbox Maildir wird oft benutzt, ist aber nicht weniger wichtig als das ältere Paradigma mbox, da es möglich sein kann, dass einige MTAs nur mbox unterstützen.

IMAP

Die Implementation einer IMAP-Überwachung ermöglicht den Zugriff auf E-Mails, die auf einem externen Mailserver gespeichert sind. IMAP ist die logische Erweiterung von POP [IMA05] und bietet erweiterte Funktionen. So ist es nicht nötig, E-Mails lokal abzuspeichern. Die benötigten Daten können synchron vom Server abgerufen und weiterverarbeitet werden.

2.5.1 Design

Es ist möglich, eines der beiden Überwachungsparadigmen zu benutzen. Wird eine lokale Überwachung vorgezogen, sollen alle lokalen Ordner gleichzeitig überwacht werden.

Lokale Überwachung

Eine Ordner- und Dateiüberwachung kann mit Hilfe des Linux-Kernel-Moduls Inotify ausgeführt werden [Lov05]. Inotify, welches mit Events über Änderungen am Dateisystem benachrichtigt, kann dabei mehrere Aufgaben zugleich übernehmen, indem ihm Überwachungsdeskriptoren hinzugefügt werden.

mbox besteht aus einer einzigen Datei, welche alle Ordner und E-Mails eines Benutzers enthält. Maildir hingegen kennt drei Ordner: Posteingang bzw. ungelesene Nachrichten, gelesene Nachrichten und den Ordner für temporäre Dateien bzw. gerade eintreffende E-Mails und speichert jede E-Mail in einer separaten Datei.

IMAP

Die Anforderungen an eine IMAP-Implementation ist, dass stets eine sichere Verbindung[6] aufgebaut werden kann (Kapitel 2.2.1) und Verbindungsabbrüche zum IMAP-

[6] In der Konfiguration des Web-Frameworks müssen die Verbindungsdaten eingetragen werden.

Server erkannt und abgefangen werden.

2.5.2 Implementation

Da sich die verschiedenen Überwachungsmaßnahmen sehr unterscheiden, muss überprüft werden, ob das System die benötigte Beschaffenheit aufweist. Wird Inotify nicht vom Kernel unterstützt, so kann keine lokale Überwachung vorgenommen werden. Desweiteren wird versucht, die Ordnerstrukturen für eine lokale Überwachung zu erstellen, falls diese nicht bereits vorhanden sind.

Lokale Überwachung

Wird die lokale Überwachung vorgezogen, werden Maildir und mbox, die Standard-Unix-Mailboxen, gleichzeitig überwacht. Dies wird mit Hilfe von pyinotify vollzogen, einer Python-Bibliothek, welche auf Inotify beruht.

- mbox wird über das Close-Event überwacht. Sobald die Datei geschlossen wurde, wird vom Notifier darüber benachrichtigt, dass eine neue E-Mail eingegangen ist. Es wäre generell möglich, auch eine Änderung der Datei zu überwachen. Dies kann allerdings Probleme verursachen, wenn der MTA mehrfach in die Datei schreibt. So würden die Funktionen, die eine E-Mail bearbeiten sollen, mehrfach aufgerufen.

  Es ist nicht möglich, die mbox-Datei des Benutzers ohne Root-Rechte zu löschen, umzubenennen oder zu ersetzen. Ebenso ist es ohne Root-Rechte nicht möglich, im mbox-Ordner neue Dateien zu erstellen. Durch die Überwachung über das Close-Event sollte die mbox-Datei nicht von anderen Programmen als dem MTA oder manuell geschlossen werden, solange diese überwacht wird. Ein Ausfall der Überwachung ist jedoch keine Alternative. Dementsprechend ist es nötig, eine Kopie dieser Datei anzulegen, um den Inhalt einer neuen E-Mail zu lesen. Diese Kopie wird in einem temporären Ordner gespeichert, auf den man Zugriffsrechte hat. Auch wenn viele E-Mails in kurzen Zeitabständen eintreffen, müssen die Dateien

eindeutig voneinander unterschieden werden. Damit Dateien nicht überschrieben werden, während sie bearbeitet werden, wird ein Dateiname vergeben, der auf der Uhrzeit basiert und Millisekunden berücksichtigt. So kann ebenso die Reihenfolge in der Warteschlange bewahrt werden. Nach der Bearbeitung werden die Kopien wieder entfernt.

- Maildir wird über das Create-Event überwacht. Der Unterordner mit neuen E-Mails bzw. Posteingang spielt die zentrale Rolle. Im Gegensatz zur Überwachung von mbox wird hierbei auf eine neu erstellte Datei reagiert. Kommt eine neue E-Mail an, oder mehrere E-Mails zugleich, werden alle Dateien im Posteingang nacheinander geöffnet. Da E-Mails im Ordner für temporäre Dateien erstellt werden und in den Posteingang verschoben werden, wird eine kurze Verzögerung implementiert, sodass auch der Posteingang überwacht werden kann aber nicht auf eine Datei zugegriffen wird, die noch nicht verschoben wurde.

Der Pyinotify-ThreadedNotifier wird gestartet und wartet auf neue E-Mails. Trifft eine Solche ein, wird der Inhalt der E-Mail an die Warteschlange weitergegeben.

mbox vs. Maildir

Obwohl mbox und Maildir lokal überwacht werden, muss man neue E-Mails mit unterschiedlichen Events behandeln.

In den Maildir-Ordnern (siehe Kapitel 2.5.1) ist es möglich, die Datei nach der Bearbeitung in den Gelesen-Ordner zu verschieben. Vorteilhaft an der einfachen Verschiebung ist, dass klar neue und gelesene E-Mails getrennt sind und man so nicht erneut auf bereits bearbeitete Daten zugreifen muss. Außerdem entfällt das nötige Kopieren und Löschen komplett (siehe Kapitel 2.5.2).

Durch diese Umstände ist es nur in Maildir möglich, viele ungelesene E-Mails auf einmal zu verarbeiten. Mit mbox kann man nur die letzte neue E-Mail erfassen. Das ist solange kein Problem, bis der Worker pausiert und man mit mbox E-Mails empfängt. Dieses geringfügige Problem ließe sich beheben, indem man die kopierte Datei nicht

löscht sondern bearbeitete E-Mails mit Gelesen-Flags versieht und zukünftig die mbox-Datei mit der Temporären verschmelzt. Dies allerdings unter der Vorraussetzung, dass die temporäre Datei weder gelöscht noch bearbeitet wird, was weitaus schlimmere Folgen haben könnte. Der programmiertechnische Aufwand wäre im Vergleich zum Nutzen also immens. Da mbox zudem ein veraltetes Paradigma für die E-Mail-Speicherung ist, empfiehlt es sich, mbox nur in Verbindung mit einem lokal installierten IMAP-Server zu überwachen und statt der lokalen Überwachung IMAP zu benutzen.

IMAP

Die Überwachung eines IMAP-Servers geschieht mit Hilfe eines Threads. Ist eine Verbindung zum IMAP-Server nicht möglich, wird sukzessiv die Zeit, in der versucht wird die Verbindung erneut aufzubauen, verdoppelt. Dies kann, je nach Anzahl der Reconnect-Versuche, von einer Minute bis hin zu 64 Minuten dauern. Danach wird die Zeit nicht mehr verdoppelt und es wird alle 64 Minuten erneut versucht eine Server-Verbindung aufzubauen. Dies passiert ebenfalls bei späteren Verbindungsabbrüchen.

Nachdem die Verbindung erstellt wurde, wartet der Thread auf ein ausgelöstes Event. Dieses Event kann durch eine neu eingetroffene E-Mail ausgelöst werden oder auch durch das Löschen einer vorhandenen E-Mail. Dabei wird nur auf Ersteres reagiert. Die bearbeiteten E-Mails erhalten automatisch das Gelesen-Flag und werden so bei zukünftigen Events nicht mehr beachtet.

Die IMAP-Überwachungsfunktion wird als Thread gestartet, sodass es möglich wäre, bei Bedarf mehrere IMAP-Server zu überwachen. Auch hier wird der Inhalt einer neuen E-Mail an die Warteschlange weitergegeben.

Lokale Überwachung vs. IMAP

Für beide Überwachungen werden zusätzliche Bibliotheken benötigt, damit in Echtzeit neu eintreffende E-Mails erfasst werden. Dabei ist die Implementation für IMAP einfacher. Im Gegensatz zur Inotify-Einbindung übernimmt der Wartevorgang der IMAP-

Überwachung ohne Voreinstellungen einen Großteil der Aufgaben, die man bei der lokalen Überwachung selbst genau spezifizieren muss.

Bei der Verwendung von IMAP ist es, wie mit Maildir, möglich, mehrere neue E-Mails zu bearbeiten; mit mbox ist dies nicht möglich.

Die Probleme mit mbox (Kapitel 2.5.2) lassen sich durch die IMAP-Überwachung umgehen. Da mit Inotify nur das Close-Event funktional sinnvoll ist und man daher nicht manuell in die User-Datei schreiben sollte, schafft ein lokal installierter IMAP-Server Abhilfe. Der IMAP-Server kann aus mbox die neuen E-Mails ohne Probleme mit Gelesen-Flags versehen, da nicht mehr auf ein Close-Event geachtet wird, sondern nur auf ein Event, das eine neu eingegangene E-Mail signalisiert.

Für IMAP spricht weiterhin das Ausbleiben von lokalen Dateien. Sämtliche Informationen werden vom Server bezogen. Bei einem Ausfall des IMAP-Servers werden nach Verbindungsneuaufbau bei einem neuen Event auch alle noch nicht bearbeiteten E-Mails nachbearbeitet, sodass ein Ausfall nur eine Zeitverzögerung aber keinen Programmausfall bedeutet.

2.6 Parsing

Oft sehen empfangene E-Mails ganz anders aus als sie verschickt werden. Nicht nur der pure Inhalt kann verändert sein, auch die Beschaffenheit einer E-Mail muss genau betrachtet werden. Viele Provider von E-Mail-Diensten fügen Signaturen in die von Ihnen verschickten E-Mails ein und es kann passieren, dass Klartext-E-Mails zu HTML-E-Mails werden. Ebenso oft werden Signaturen von den Benutzern selbst benutzt, z.B. Abwesenheitsnotizen oder auch PGP-Signaturen. Das Parsing wird zur unabdingbaren Funktion für eine Gewissheit, dass dem Kommentar weder überflüssige Informationen hinzugefügt werden, noch ein Informationsverlust eintritt.

2.6.1 Design und Implementation

Das Parsing muss Multipart-E-Mails unterstützen. Klartext- und HTML-Teile sollen als formatierter Text zurückgegeben werden. Ebenso können angehängte Bilder zurückgegeben werden. Diese sollen in den Kommentar eingefügt werden. Eine weitere Anforderung ist das Entfernen jeglicher Art von Signatur und eine Möglichkeit, zu erkennen, ob der Benutzer ein Votum abgeben möchte. Insbesondere wird hier auch die Empfängeradresse analysiert.

Um das Parsing auf Funktionalität hin zu untersuchen wurde ein Test-Modul geschrieben.

Die Warteschlange ruft das Parsing auf, sobald ein neuer Auftrag bearbeitet werden soll. Zuerst wird versucht, aus der Empfängeradresse die benötigten Daten zu gewinnen. Danach wird der lesbare Inhalt inklusive der angehängten Bilder extrahiert. Ebenso wird das Votum, falls vorhanden, übergeben. Sollte der Kommentar weniger als 4 Zeichen lang sein, wird wieder eine Benachrichtigung an den Benutzer verschickt, dass sein Kommentar nicht bearbeitet werden konnte, weil er zu kurz ist.

Empfängeradresse

Zuerst wird die Beschaffenheit und der Inhalt der Empfängeradresse[7] analysiert.

Der lokale Teil der E-Mail-Adresse wird dann auf die erwähnte Form (siehe Kapitel 2.2.2) hin überprüft. Bei korrekter Form werden wieder Benutzer-ID, Kommentar-ID und Crypto-Secret aneinandergereiht und gehasht. Dieser Wert wird mit dem übergebenen Sicherheitstoken aus der Empfängeradresse verglichen. Außerdem muss überprüft werden, ob Benutzer-ID und Kommentar-ID existieren. Dazu wird in der Datenbank nach den IDs gesucht und, falls vorhanden, werden die jeweiligen Objekte zurückgegeben. Diese Objekte beinhalten spezifische Informationen zu Benutzer und Kommentar.

Sollte es nicht möglich sein, die Empfängeradresse zu überprüfen, wird eine Benach-

[7] Es erfolgt eine Umwandlung von *Name* <*name@domain.tld*> zu *name@domain.tld*.

richtigung darüber an den Benutzer verschickt. Es kann mehrere Gründe geben, dass eine Prüfung fehlschlägt. Einerseits kann die E-Mail-Adresse eine falsche Form haben. Andererseits kann die Überprüfung über Existenz von Benutzer und Kommentar fehlschlagen und dadurch auch das Sicherheitstoken falsch sein. Ebenso kann es auch sein, dass nur das Sicherheitstoken falsch angegeben wurde. In jedem dieser Fälle liegt die Vermutung nahe, dass der Benutzer die Empfängeradresse abgeändert hat. Wird allerdings z.B. der Server neu gestartet, wird das Crypto-Secret neu generiert. Dadurch verlieren alle bis dahin verschickten Sicherheitstokens ihre Gültigkeit. Man müsste für eine Lösung dieses Problems ein Secret persistent speichern. Z.B. könnte man dieses Secret in der Datenbank speichern.

Payload

Zuerst muss überprüft werden, ob es sich um eine Multipart-E-Mail handelt. Diese können Bilder oder HTML enthalten. Andernfalls ist Klartext enthalten. Im E-Mail-Header kann am *Content-Type* abgelesen werden, um welchen MIME-Typen es sich handelt. Die lesbaren Formate werden aneinandergereiht[8], wenn eine so beschaffene E-Mail eingetroffen ist. Aktuell werden HTML- und Klartextnachrichten unterstützt. Keine Aneinanderreihung findet statt, wenn es sich um dem MIME-Typen *multipart/alternative* handelt. Dabei werden mehrere Versionen des gleichen Texts angeboten. Meist findet man in so einer E-Mail eine Klartext- und eine HTML-Nachricht vor. Dabei wird nur eine Version geparsed. HTML-Text wird bevorzugt behandelt, weil dort Formatierungen gespeichert sind, die man in die Kommentarantwort übernehmen kann. Die Klartext-Alternative bietet diese Möglichkeit nicht.

Multipart-Messages

Für das Parsing von Multipart-Messages werden zuerst sämtliche Payloads und deren Content-Type zwischengespeichert. Außerdem wird gespeichert, ob es sich um alternative Versionen handelt. Damit zwischen mehreren *multipart/alternative*-Teilen unter-

[8] Der Payload wird bei den MIME-Typen *text/plain* und *text/html* in Unicode gespeichert.

schieden werden kann, wird zudem zwischengespeichert, ob ein Zusammenhang zwischen mehreren Alternativen besteht. Hier wird sich die Reihenfolge zunutze gemacht. Alternativen treten immer nacheinander auf. Ebenso könnte man Alternativen anhand der angegebenen Grenzen (Boundaries) im Payload analysieren.

Besteht der Payload aus HTML-Text, muss dieser zu Klartext umgewandelt werden. Hier wird außerdem die spezielle, einfach strukturierte Markdown-Syntax, die Kommentare formatiert, benutzt. Sind Fettdruck-, Kursivschrift-, Überschrift-, Listen-, Weblink-, Bildverweis- und Zitat-Tags in der HTML-E-Mail vorhanden, werden diese in die Markdown-Syntax umgewandelt. Markdown wandelt zwar seine Syntax wieder zu HTML um, sobald Kommentare auf der Webseite angezeigt werden sollen, eine direkte Benutzung von HTML ist aber aus Sicherheitsgründen nicht möglich. Würde HTML erlaubt werden, könnten z.B. schädliche JavaScript-Funktionen eingebunden werden und das Verhalten und Aussehen der Web-Seite beeinträchtigt werden.

Damit eingehende HTML-E-Mails dennoch bearbeitet und die Formatierung so weit wie möglich beibehalten werden kann, wird eine Parsing-Funktion implementiert, die HTML-E-Mails in Markdown-Syntax umwandelt.
Leerzeichen nach Schlüsselbegriffen bzw. -zeichen der Markdown-Syntax müssen genau beachtet werden. Diese sind in Tabelle 2.1 unterstrichen. Zu beachten ist, dass Listen bisher nur ungeordnet und ohne Einrückung implementiert sind. Weiterhin enthalten Web-Links und Bild-Referenzen ein Duplikat der URL. Die Angabe ist zwar nicht zwingend erforderlich, bietet bei einem Mouseover der Web-Links aber die Möglichkeit, die URL des Web-Links beim Maus-Cursor anzuzeigen. Das Duplikat der URL der Bild-Referenz hingegen stellt den Bildtitel dar. Im Internet-Explorer hat der Bildtitel Vorrang gegenüber dem alt-Attribut und dient wiederum als Tooltip [Bil07]. Alle HTML-Tags, die kein Markdown-Äquivalent besitzen, werden ersatzlos entfernt.

Damit Bilder, die an die E-Mail angehängt wurden, auf der Webseite sichtbar sind, muss beim Rendering-Vorgang bisher der *Safe-Mode* von Markdown abgestellt werden. Es wurde ein Pull-Request für die Python-Implementation von Markdown erstellt, sodass in Zukunft das Darstellen von Data-URIs evtl. möglich sein wird. Die Data-URI ersetzt im Bild-Tag den http-Verweis. Dadurch, dass Data-URIs allerdings durch die vielfälti-

HTML	Markdown-Syntax
 	_\n
Fettdruck	**Fettdruck**
Fettdruck	
Kursivschrift	*Kursivschrift*
Kursivschrift	
<q>Zitat</q>	\n> Zitat_\n\n
<blockquote>Zitat</blockquote>	
H1-Überschrift	# H1-Überschrift
...	...
H6-Überschrift	###### H6-Überschrift
 Beschreibung	[Beschreibung] (http://www.example.com "http://www.example.com")
<img src="www.example.com/ex.jpg" alt="Beschreibung">	![Beschreibung] (http://www.example.com/ex.jpg "http://www.example.com/ex.jpg")
Listenelement1 Listenelement2	* Listenelement1_\n * Listenelement2_\n

Tabelle 2.1: Beispielumwandlungen von HTML zu Markdown-Syntax

gen Möglichkeiten Schaden anrichten können, empfiehlt der Entwickler von Markdown statt der Implementation von Data-URIs den Safe-Mode für bestimmte Zwecke zu deaktivieren [Lim13]. Durch Data-URIs ist es wieder möglich, HTML einzubetten. Dadurch könnte auch JavaScript wieder ausgeführt werden, was die Web-Seite oder Benutzer schädigen könnte (siehe auch Kapitel 2.6.1). Um dieses Problem zu vermeiden, müsste der Safe-Mode von Markdown in der Lage sein, zwischen den MIME-Typen zu unterscheiden, die in einem Data-URI angegeben werden. Da eine Implementation weiterer Funktionen für den Safe-Mode allerdings nicht geplant ist, muss entweder, wie vom Entwickler von Markdown empfohlen, die API von Markdown benutzt werden, um das bisherige Verhalten des Safe-Modes aufzuheben bzw. zu ändern oder eine Änderung in adhocracy vorgenommen werden. Durch die bisherige Implementation in adhocracy gestaltet sich ein selektives Deaktivieren des Safe-Modes schwierig. Eine komplette Deaktivierung hätte zur Folge, dass HTML wieder erlaubt wird. Denkbar ist die Benutzung einer zusätzlichen Python-Bibliothek für den Zweck, HTML zu verbieten. Auch könnte

eine angepasste Version von Markdown verwendet werden, die den Safe-Mode so implementiert, dass Data-URIs für Bilder erlaubt sind.

Votevorgang

Es kann eine von vier verschiedenen Stimmabgaben (Vota) gewählt werden:

- positives Votum
- negatives Votum
- neutrales Votum
- kein Votum

Der Benutzer kann in der ersten Zeile seiner Kommentarantwort ein Votum über den Kommentar, auf den er antwortet, abgeben. Er hat die Möglichkeiten, diese Zeile durch `Vote` oder `vote` zu kennzeichnen. Dabei können danach auch ein Doppelpunkt und beliebig viele folgende Leerzeichen benutzt werden. Es ist auch möglich, das Votum direkt anzugeben: Ein positives Votum kann danach durch +, +1 oder 1 angegeben werden. Für die Datenbank wird das Votum zu „1“ umgewandelt. Für ein negatives Votum besteht die Möglichkeit, - oder -1 anzugeben. Hier wird zu „-1“ umgewandelt. Ein neutrales Votum kann mit der Angabe von 0 gekennzeichnet werden; möchte man kein Votum abgeben, kann die erste Zeile direkt für die Kommentarantwort benutzt werden. Wird allerdings eine Kommentarantwort, z.B. mit „1“ eingeleitet und danach in der selben Zeile Text geschrieben, wird dies nicht als Votum erkannt. Es kann sein, dass dies Bestandteil der Antwort des Benutzers ist.
Wird ein Votum abgegeben, wird dieses gespeichert und die erste Zeile mitsamt aller vor der eigentlichen Kommentarantwort auftretenden Leerzeilen gelöscht. Die Benachrichtigung über neue Kommentare beinhaltet bereits die Vote-Zeile. So wird kenntlich gemacht, dass die Funktion vorhanden ist. Eine Erklärung zum Vote-Vorgang wurde ergänzend hinzugefügt. Ebenso wird die Vote-Zeile vom Text der Benachrichtigung so abgetrennt, dass klar ist, dass darunter die eigentliche Antwort Platz findet.

Es ist noch nicht möglich, nur ein Votum ohne Antwort abzugeben. Durch die minimale Länge von 4 Zeichen eines Kommentars ist es bisher noch erforderlich, eine Antwort zu schreiben.

Signaturen

Signaturen stehen am Ende einer E-Mail. Bei Multipart-E-Mails werden auftretende Signaturen pro Teil entfernt. Standardmäßig werden Signaturen mit „−−“ am Anfang einer Zeile eingeleitet [Gel04]. Es gibt allerdings E-Mail-Provider, die ihre Signaturen mit „___“ einleiten. Auch PGP-Signaturen unterliegen diesem Standard. Der eigentliche Payload wird allerdings zusätzlich gekennzeichnet. Über diesem Payload fügt PGP die Zeile *−−−−−BEGIN PGP SIGNED MESSAGE−−−−−* hinzu. In der folgenden Zeile wird der Hash-Header angegeben und noch eine Leerzeile hinzugefügt [CDF+07]. Diese Informationen werden ebenfalls entfernt. Dies geschieht, bevor auf die eigentlichen Signaturen hin überprüft wird, da bei einer erkannten Signatur der restliche Text gelöscht wird. Nach Entfernen der Signaturen werden außerdem alle führenden und nachfolgenden Leerzeichen und Leerzeilen entfernt.

Die bereits erwähnte Vorformulierung der Benachrichtigungen (siehe Kapitel 2.6.1), wurde so angepasst, dass die bisherigen Informationen über den geschriebenen Kommentar als Signatur erkannt werden. Dieser Teil muss nicht mehr gelöscht werden und der Benutzer kann jetzt direkt nachdem er eine Antwort geschrieben hat, die E-Mail versenden. Manche MUAs setzen standardmäßig den Cursor unter die empfangene E-Mail, wenn man antworten möchte. Es befindet sich am Ende der vorformulierten Benachrichtigung ein Hinweis darauf, dass man über der Linie, die die Signatur einleitet, antworten muss. Auch die E-Mail-Kommentar-Funktion von Facebook ist nicht in der Lage, unterhalb der vorformulierten Benachrichtgung einen Kommentar als solchen zu erkennen.

Parsing-Tests

Anhand von Beispiel-E-Mails, die echten E-Mails ähneln und verschiedener, oft auftretender Unterschiede in Formatierungen (z.B. \r\n oder \n), wurden Tests durchgeführt.

Da Data-URIs bisher nicht gerendert werden können, schlägt ein solcher Test fehl. Erfolgreich hingegen verlaufen HTML-Tests, das Parsen des Votums, das Entfernen sämtlicher Signaturen, das Lesen des Content-Type und die Analyse der E-Mail-Adresse des Empfängers.

2.7 Datenbankanbindung

Um eingegangene E-Mail-Kommentarantworten dauerhaft zu speichern, wird eine relationale Datenbank benutzt. Mit der Datenbanksprache SQL und mit Hilfe der WSGI-Middleware SQLAlchemy werden Aufträge an die Datenbank bearbeitet.

2.7.1 Design

Es wird eine Datenbanksession benötigt, die alle Daten lagert, die in die Datenbank geschrieben werden sollen. Für diesen Zweck wird eine Scoped-Session erstellt. Sie arbeitet thread-lokal und hat den Vorteil, dass andere Threads die gelagerten Daten nicht verändern oder an falscher Stelle verwenden können [Ses13].

Für eine erfolgreiche Verbindung zur Datenbank, mit der Absicht eine Kommentarantwort zu speichern, benötigt adhocracy außerdem einen eingerichteten Template-Context und eine Schnittstelle zur Autorisierung (siehe Kapitel 2.2.1). Ebenfalls müssen für die Einrichtung des Template-Contexts Instanz-Filter und Request konfiguriert bzw. erstellt werden.

Verhindern von Kommentarduplikaten

In einer Produktionsumgebung kann es sein, dass mehrere Worker gestartet werden. Es ist also möglich, dass die selben E-Mail-Ordner mehrfach überwacht werden. Bei der IMAP-Überwachung kann z.B. eine Verzögerung beim Setzen der Gelesen-Flags entstehen. In dieser Zeit könnte bereits ein weiterer Worker auf diese E-Mail zugreifen. Um

zu verhindern, dass die selbe E-Mail mehrfach in die Datenbank geschrieben wird, muss eine Überprüfung stattfinden, ob diese E-Mail bereits bearbeitet wurde. Falls dem so ist, soll keine Verbindung zur Datenbank aufgebaut werden.

Es gibt mehrere Möglichkeiten, eine Implementation vorzunehmen:

- Eine **Prüfung von Duplikaten** kann einfach implementiert werden. Es wird dazu in der Datenbank nach bereits geschriebenen Kommentaren des Benutzers gesucht. Da man Duplikate nicht generell verbieten möchte, wird dazu eine geringe Zeitspanne betrachtet. Es wird geprüft, ob ein Kommentar denselben Inhalt besitzt und ob er auf den gleichen Kommentar antworten soll. Mit Hilfe der betrachteten Zeitspanne kann man Duplikate schnell feststellen und man verhindert innerhalb dieser Zeitspanne außerdem sogenannte Doppelposts.
  Gleichzeitig ist die Benutzung einer festen Zeitspanne nachteilig. Wie viele Worker Aufträge bearbeiten und wie lange benötigt wird, die jeweilige Warteschlange abzuarbeiten, wird nicht betrachtet. Die Zeitspanne sollte so hoch gesetzt werden, dass die Worker bereits die E-Mail verarbeiten konnten und ebenso so niedrig, dass wieder Antworten auf denselben Kommentar vom Benutzer eingehen können, die auch denselben Text beinhalten.

- Alternativ kann eine **persistente Speicherung des Empfangsdatums der E-Mail** implementiert werden. Mit dieser Methode kann sicher überprüft werden, ob eine E-Mail bereits bearbeitet wurde. Somit wird eine redundante Speicherung effektiv unterbunden. Die Speicherung könnte in der Datenbank erfolgen. Sinnvoll wäre ein zusätzliches Feld bzw. eine zusätzliche Spalte in der Tabelle, in der die Kommentare gespeichert sind.

- Ebenso kann man die **Worker untereinander synchronisieren**, sodass jeder Worker von den anderen Workern weiß, welche E-Mail er bearbeitet. Diese E-Mails könnten für andere Worker gesperrt werden, bis sie als gelesen markiert sind. Dadurch würde ein redundantes Öffnen einer E-Mail verhindert und es würde ein schnelleres Abarbeiten der Warteschlangen ermöglicht.

Instanz-Filter

Ein Vorschlag befindet sich immer in einer Instanz. Eine Instanz stellt hierbei das Hauptthema aller Vorschläge dar. Dem jeweiligen Thread muss zwecks Autorisierung diese Instanz erst zugewiesen werden.

Request

Da das Schreiben in die Datenbank benutzerabhängig ist, wird eine Anfrage bzw. eine (HTTP-)Request erstellt. Als Grundgerüst wird eine leere Request erzeugt, in die die entsprechenden Berechtigungen des Benutzers geladen werden.

Template-Context

Der Template-Context beinhaltet Request-spezifische Daten und überträgt diese in ein Template [BGJ13].

Hier werden benutzerspezifische Daten und die eingerichtete Instanz gespeichert. Außerdem muss eine Verbindung zum Indizierungsdienst angegeben werden, damit dieser bei einem neuen Kommentar die interne Suche aktualisiert.

2.7.2 Implementierung

Nach Parsing der E-Mail wird der Kommentar darauf vorbereitet, in die Datenbank geschrieben zu werden. Zuerst wird geprüft, ob der selbe Kommentar bereits bearbeitet wurde und schon in der Datenbank vorhanden ist. Bisher ist aus Zeitgründen eine Implementation über die Prüfung von Duplikaten mit fester Zeitspanne der Worker-Synchronisation vorgezogen worden. Die Zeitspanne beträgt zwei Minuten.
Dann wird via Request überprüft, ob der Benutzer autorisiert ist, eine Antwort auf den angegebenen Kommentar zu verfassen. Sollte er nicht autorisiert sein, wird er per E-Mail

darüber benachrichtigt, dass der Versuch, eine Antwort zu schreiben, fehlgeschlagen ist. Ist der Benutzer hingegen berechtigt, wird der Kommentar mit folgenden Attributen erstellt:

- aus Parsingvorgang übergebene kommentarspezifische Daten
- aus Parsingvorgang übergebene benutzerspezifische Daten
- Vorschlag, in dem sich der Kommentar befindet
- Payload der E-Mail
- wiki des Kommentars
- Variante des Kommentars
- vom Benutzer abgegebenes Votum

Das Wiki ist die Eigenschaft eines Kommentars, Veränderungen am eigenen Kommentar von anderen Benutzern zuzulassen. Das Wiki wird standardmäßig auf 0 gesetzt. Damit wird verhindert, dass andere Benutzer den Kommentar bearbeiten können. Die Variante des Kommentars bezeichnet eine bestimmte Version des Vorschlags. Es ist möglich, dass Vorschläge durch abgegebene Vota abgeändert werden und andere Versionen entstehen.

Request, Template-Context und Instanzfilter werden nach dem Kommentiervorgang gelöscht bzw. geleert.

Kapitel 3

Resümee

Ziel der Arbeit war es, unter Beachtung von Sicherheitsaspekten, eDemocracy-Systeme so zu erweitern, dass die Möglichkeit besteht, Antworten auf Kommentare per E-Mail zu verfassen. Zu diesem Zweck wurde beschrieben, wie ein MTA eingerichtet sein muss und eine Implementation in das eDemocracy-System adhocracy vorgenommen.

Für die Implementation wurde ein neuer Worker eingerichtet. Der Worker übernimmt mit einer Warteschlange sämtliche Aufgaben der E-Mail-Kommentare: Das Feststellen des Empfangs einer neuen E-Mail in lokalen Ordnern oder über IMAP, das Abholen und Verarbeiten des Inhalts bis hin zum Einfügen in die Datenbank.

Die Implementation in adhocracy bietet die Möglichkeit, Bilder anzuhängen und zu verlinken. Dazu kann ebenso HTML genutzt werden, ohne ein Sicherheitsrisiko darzustellen. Weiterhin ist es möglich, ein Votum abzugeben. Auch ist die Unabhängigkeit von der Absender-E-Mail-Adresse des Benutzers gegeben. Essentiell erschien außerdem das Entfernen aller Signaturen von Benutzer oder E-Mail-Provider.

Es ist möglich, dieselben E-Mail-Ordner simultan von mehreren Workern überwachen zu lassen, ohne dass eine Kommentarantwort mehrfach bearbeitet wird.

Die Möglichkeit, plattform- bzw. browser-unabhängig und damit auch mobil am adhocracy-Kommentar-System teilnehmen zu können, ist durch die Integration der E-Mail-Kommentar-Funktion sichergestellt.

Kapitel 4

Ausblick

Es gibt viele Möglichkeiten, die Funktionen der Implementierung noch zu erweitern. Durch die Modularisierung des Programms ist es sehr einfach, zusätzliche Funktionen hinzuzufügen.

4.1 Überwachung

4.1.1 Überwachung von Nicht-Standard-Mailboxen

Die Überwachungsfunktionen können dahingehend ergänzt werden, dass Nicht-Standard-Mailboxen überwacht werden können. Ebenso aber auch der Spool-Ordner, der einen Überordner von mbox darstellt und häufig von MTAs verwendet wird. Es erscheint sinnvoll, die Mailbox-Ordner in der Konfigurationsdatei angeben zu können.

4.1.2 Kommentarduplikate

Die Funktion, die Kommentarduplikate unterbinden soll, basiert auf einer fest eingestellten Zeitspanne. Diese Funktion sollte anhand der genannten Alternativen (siehe Kapitel 2.7.1) geändert bzw. erweitert werden.

4.1.3 Lokale Überwachung unter BSD-Derivaten

Durch die Implementation von Pyinotify ist eine Verwendung der lokalen Überwachung unter BSD-Derivaten oder Mac OS X nicht möglich, da das Kernelmodul Inotify dort nicht vorhanden ist. Allerdings gibt es andere Möglichkeiten, das Dateisystem auf Änderungen hin zu überprüfen. Die Python-Bibliothek watchdog [Man12] ist fähig, Dateisystemänderungen unter z.B. Mac OS X zu erkennen. Dabei ist es möglich, watchdog auch für eine Überwachung anhand von Inotify zu benutzen. Somit wäre es evtl. möglich, Pyinotify komplett zu ersetzen.

4.1.4 Initiales Abholen von neuen E-Mails

Bei einem Dienstausfall der Überwachung können bei IMAP und Maildir bereits E-Mails bearbeitet werden, die in der Zeit des Ausfalls eingingen. Dies ist allerdings erst möglich, sobald ein neues Event dazu anregt, neue E-Mails zu überprüfen. Bei einem Neustart bzw. einer Wiederaufnahme des Dienstes ist es sinnvoll, initial zu überprüfen, ob neue E-Mails empfangen wurden.

4.1.5 Alternative IMAP-Server

Es sollte für den Fall, dass ein IMAP-Server ausfällt oder gewartet werden muss, eine Funktion implementiert werden, die fähig ist, zur Laufzeit auf einen anderen IMAP-Server zu wechseln und diesen zu überwachen. Dazu müsste es möglich sein, in der Konfigurationsdatei Benutzerdaten für alternative IMAP-Server anzugeben.

4.2 Parsing

4.2.1 HTML-E-Mails

Das Parsing sollte in Zukunft in der Lage sein, zwischen der eigentlichen Kommentarantwort und dem Inhalt der Benachrichtigung über neue Kommentare zu unterscheiden. Dies gilt insbesondere für Multipart-Benachrichtigungen. Dies eröffnet die Möglichkeit, HTML-E-Mails zu verschicken. Durch den Multipart-Charakter von HTML-E-Mails könnte es bisher passieren, dass die Benachrichtigung mitgeparsed wird, falls der Benutzer sie nicht löscht. Das Grundgerüst für das Verschicken von HTML-E-Mails ist bereits integriert, kann aber aufgrund der genannten fehlenden Funktionen bisher noch nicht eingesetzt werden.

4.2.2 Absatzbasierte Antworten

Das Parsing kann außerdem durch die Möglichkeit erweitert werden, Textabsätze zu erkennen. Eine Einbindung von Antworten, die man in Absätze unterteilen kann, böte die Möglichkeit, bei langen Kommentaren übersichtlicher auf einzelne Absätze zu antworten. Dabei könnten Kommentare auf der Webseite durch Verweise oder Überschriften gekennzeichnet sein. Ebenfalls ist denkbar, dass Antworten auf bestimmte Absätze an betreffender Stelle gerendert werden. Sinnvoll wäre hier eine Implementation zum Ein- und Ausblenden von Antworten. Eine aufklappbare Miniaturübersicht der Kommentare und Antworten zu Absätzen könnte ebenfalls zur Lesbarkeit beitragen.

4.2.3 Subjekt der E-Mail

Weiterhin könnte analysiert werden, ob das bisher ungenutzte Subjekt einer E-Mail benutzt werden könnte. Das Subjekt könnte für eine Überschrift innerhalb des Kommentars oder für das Voting benutzt werden. Insbesondere bei Multipart- bzw. HTML-E-Mails wäre Letzteres sinnvoll, da in HTML-E-Mails Zeilen generell mit Tags beginnen. Bis-

her ist die Voraussetzung für das Voting die Nutzung der ersten Zeile des Payloads der E-Mail. Auch könnte hier als zusätzliche Funktion das Wiki des Kommentars (siehe Kapitel 2.7.2) implementiert werden. Bisher ist es nicht möglich, dass E-Mail-Kommentare von anderen Benutzern direkt bearbeitet werden können.

4.2.4 Voting

Es sollte in Zukunft möglich sein, ein Votum abzugeben, auch ohne dass dazu ein Kommentar verfasst werden muss (siehe Kapitel 2.6.1).

4.3 PGP-Einbindung

Eine Erweiterung für eine noch höhere Sicherheit stellt eine PGP-Einbindung dar. Durch die Verwendung von PGP kann dem Benutzer ermöglicht werden, verschlüsselte E-Mails zu verschicken. Dazu kann vom Benutzer ein öffentlicher Schlüssel von adhocracy benutzt werden. Mit dem privaten Schlüssel von adhocracy kann die E-Mail serverseitig entschlüsselt werden.

4.4 Crypto-Secret

Das Crypto-Secret, das benutzt wird, um ein Sicherheits-Token zu erstellen, wird nach einem Server-Neustart geändert. Dadurch verlieren die generierten Sicherheits-Tokens ihre Gültigkeit. Das Crypto-Secret sollte in Zukunft entweder persistent gespeichert werden oder es sollte ein anderes Secret benutzt werden, was wiederum persistent gespeichert werden müsste.

4.5 E-Mail-Benachrichtigungen

E-Mail-Benachrichtigungen über neue Kommentare werden für eine direkte Kommentarantwort bereits vorformatiert. Ebenso wird eine Antwortadresse generiert. Die Erklärungen in dieser Benachrichtigung sollen dem Benutzer ermöglichen, die vorhandenen Funktionen effektiv zu benutzen. Der Worker, der diese Funktionen ermöglicht, kann deaktiviert werden, wenn man den Dienst nicht nutzen möchte. Die Benachrichtigungen sollten in Zukunft nur vorformatiert verschickt werden, wenn der Worker auch aktiviert ist. Durch die Vorformatierung der Benachrichtigungen wird sonst fälschlicherweise suggeriert wird, dass der Dienst aktiv ist.

Literaturverzeichnis

[BGJ13] BANGERT, B. ; GARDNER, J. ; JENVEY, P.: *Passing Variables to Templates*. http://pylons-webframework.readthedocs.org/en/v0.9.7/views.html. Version: Januar 2013 (Pylons Documentation)

[Bil07] SELFHTML: *Grafiken einbinden*. de.selfhtml.org/html/grafiken/einbinden.htm#referenz. Version: 2007

[CDF+07] CALLAS, J. ; DONNERHACKE, L. ; FINNEY, H. ; SHAW, D. ; THAYER, R.: *OpenPGP Message Format*. RFC 4880 (Proposed Standard). http://www.ietf.org/rfc/rfc4880.txt. Version: November 2007 (Request for Comments). – Updated by RFC 5581

[Dri13] DRIESSEN, V.: *Workers*. http://python-rq.org/docs/workers/. Version: 2013 (RQ Documentation)

[FKK11] FREIER, A. ; KARLTON, P. ; KOCHER, P.: *The Secure Sockets Layer (SSL) Protocol Version 3.0*. RFC 6101 (Historic). http://www.ietf.org/rfc/rfc6101.txt. Version: August 2011 (Request for Comments)

[Gel04] GELLENS, R.: *The Text/Plain Format and DelSp Parameters*. RFC 3676 (Proposed Standard). http://www.ietf.org/rfc/rfc3676.txt. Version: Februar 2004 (Request for Comments)

[IMA05] Universität Innsbruck: *IMAP- und POP-Mail: Unterschiede und Konzepte*. www.uibk.ac.at/zid/systeme/mail/imappop.html. Version: Dezember 2005 (mail)

[Kam12] KAMP, P.-H.: *Md5crypt Password scrambler is no longer considered safe by author*. `phk.freebsd.dk/sagas/md5crypt_eol.html`. Version: Juni 2012

[Kle08] KLENSIN, J.: *Simple Mail Transfer Protocol*. RFC 5321 (Draft Standard). `http://www.ietf.org/rfc/rfc5321.txt`. Version: Oktober 2008 (Request for Comments)

[Lim13] LIMBERG, W.: *Pull-Request-Comment*. `https://github.com/waylan/Python-Markdown/pull/235`. Version: August 2013

[Lov05] LOVE, R.: *inotify - a powerful yet simple file change notification system*. `https://github.com/lyfkevin/MIUIv5_iproj_kernel/blob/master/Documentation/filesystems/inotify.txt`. Version: März 2005

[Man12] MANGALAPILLY, Y.: *watchdog package documentation*. `https://pypi.python.org/pypi/watchdog`. Version: März 2012

[Res01] RESNICK, P.: *Internet Message Format*. RFC 2822 (Proposed Standard). `http://www.ietf.org/rfc/rfc2822.txt`. Version: April 2001 (Request for Comments). – Obsoleted by RFC 5322, updated by RFCs 5335, 5336

[Ses13] SQLAlchemy authors and contributors: *Using the Session*. `http://docs.sqlalchemy.org/en/rel_0_8/orm/session.html`. Version: Juli 2013 (SQLAlchemy 0.8 Documentation)

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 12. August 2013 Denis Klaus Küchemann

Diese CD enthält:

- eine *pdf*-Version der vorliegenden Bachelorarbeit
- die LaTeX- und Grafik-Quelldateien der vorliegenden Bachelorarbeit samt aller verwendeten Skripte
- die Quelldateien der im Rahmen der Bachelorarbeit erstellten Software
- die Websites der verwendeten Internetquellen